Cryptography and algorithmic randomness II

— The Generic Group Model and Effective Hardness —

Kohtaro Tadaki

Research and Development Initiative, Chuo University
Tokyo, Japan

Supported by KAKENHI (23340020), Japan Society for the Promotion of Science

Abstract

In modern cryptography, the generic group model (Shoup, 1997) is widely used as an *imaginary* framework in which the security of a cryptographic scheme is discussed.

In particular, the generic group model is often used to discuss the computational hardness of problems, such as the discrete logarithm problem and the Diffie-Hellman problem, which are used as a computational hardness assumption to prove the security of a cryptographic scheme.

In this talk, we apply the concepts and methods of *algorithmic randomness* to the generic group model, and consider the secure instantiation of the generic group, i.e., a random encoding of the group elements.

In particular, we show that the generic group can be instantiated by a specific computable function while keeping the computational hardness of the problems originally proved in the generic group model.

Abstract

In CCR 2012, we considered the secure instantiation of the random oracle. Here, the random oracle model is more widely used than the generic group model as an imaginary framework in which the security of a cryptographic scheme is discussed.

In this talk, we show that the same line of research is possible for the generic group model.

Computational Hardness Assumptions

Computational Hardness Assumptions about Groups

There are several computational hardness assumptions with respect to finite cyclic groups to prove the security of cryptographic schemes.

- The hardness of *the discrete logarithm problem*
- The hardness of *the computational Diffie-Hellman problem*
- The hardness of *the decisional Diffie-Hellman problem*
- .....

Computational Hardness Assumptions about Groups

There are several computational hardness assumptions with respect to finite cyclic groups to prove the security of cryptographic schemes.

- The hardness of *the discrete logarithm problem*
- The hardness of *the computational Diffie-Hellman problem*
- The hardness of *the decisional Diffie-Hellman problem*
- .....

The Discrete Logarithm Problem

Finite Cyclic Groups

- A group G is called *cyclic* if there exists $g \in G$ such that

$$G = \{g^i \mid i \in \mathbb{Z}\}.$$

Such g is called a *generator* of G.

- The number of elements in a finite group G is called the *order* of G.

- For every finite cyclic group G and every generator g of G,

$$G = \{g^0, g^1, \ldots, g^{m-1}\},$$

where m is the order of G.

Thus, G is isomorphic to the *additive* group $\mathbb{Z}_m$ by

$$G \ni g^i \mapsto i \in \mathbb{Z}_m,$$

where $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$ with the binary operation $\circ$ for $a_1, a_2 \in \mathbb{Z}_m$ defined by

$$a_1 \circ a_2 := (a_1 + a_2) \bmod m.$$

Finite Cyclic Groups

Example

Let p be a prime. Consider the set

$$\mathbb{Z}_p^* := \{a \in \mathbb{Z}_p \mid \gcd(a, p) = 1\} = \{1, 2, \ldots, p-1\}.$$

This set is a group with the binary operation $\circ$ for $a_1, a_2 \in \mathbb{Z}_p^*$ defined by

$$a_1 \circ a_2 := a_1 a_2 \bmod p.$$

The group $\mathbb{Z}_p^*$ is shown to be a finite cyclic group of order $p-1$. We also see that there are $\phi(p-1)$ generators of G, where ϕ is the Euler function defined by

$$\phi(N) := \#\{a \in \mathbb{Z}_N \mid \gcd(a, N) = 1\}.$$

□

Discrete Logarithm

Definition Let G be a finite cyclic group of order q and g its generator. Then, for every $h \in G$ there is a *unique* $x \in \mathbb{Z}_q$ such that $g^x = h$. We call this x the *discrete logarithm of h with respect to g* and write

$$x = \log_g h.$$

□

Discrete logarithms obey many of the same rules as "standard" logarithms. For example,

(i) $\log_g 1 = 0$, where 1 is the unit element of G,

(ii) $\log_g(h_1 h_2) = (\log_g h_1 + \log_g h_2) \bmod q$.

The discrete logarithm problem is to find the discrete logarithm $\log_g h$, given a generator g of G and an element $h \in G$. The hardness of the discrete logarithm problem is the hardness to find the discrete logarithm.

Experiment for the Discrete Logarithm Problem

Let G be a finite cyclic group in a certain class.

Consider the following experiment defined for a probabilistic polynomial-time algorithm $\mathcal{A}$ and a parameter n:

The discrete logarithm experiment $\mathsf{DLog}_{\mathcal{A}}(n)$**:**

1. *Generate (G, q, g), where G is a finite cyclic group of order q represented by n bit strings and g is a generator of G.*
2. *Generate $h \in G$ uniformly.*
3. *$\mathcal{A}$ is given q, g, h and outputs $x \in \mathbb{Z}_q$*
4. *The output of the experiment is defined to be 1 if $g^x = h$ and 0 otherwise.*

The Hardness of the Discrete Logarithm Problem

Definition

We say that *the discrete logarithm problem is hard* (with respect to a certain class of finite cyclic groups) if for all probabilistic polynomial-time algorithms $\mathcal{A}$ and all $d \in \mathbb{N}^+$ there exists $N \in \mathbb{N}^+$ such that, for all $n > N$,

$$\mathsf{Prob}[\mathsf{DLog}_{\mathcal{A}}(n) = 1] \leq \frac{1}{n^d}.$$

□

The hardness of the discrete logarithm problem is one of the major computational hardness assumptions by which the security of cryptographic schemes is proved.

The Generic Group Model

The Generic Algorithm

The Generic Algorithm: Motivation and Intuition

We want to consider group algorithms which only use the minimal properties of group as a finite cyclic group.

The generic algorithms are *generic* group algorithms in the sense that they apply equally well to all finite cyclic groups. The generic algorithms do not relay on specific properties of a *particular* finite cyclic group or class of finite cyclic groups.

To realize this, the group operations of a finite cyclic group are performed via oracle calls by the generic algorithms, and all possible finite cyclic groups of a given order are considered as an oracle in a randomized manner.

Thus, Shoup introduced the notion of generic algorithm in 1997.

Encoding Function into n Bitstrings

Definition [Encoding Function into n Bitstrings]
Let $n \in \mathbb{N}^+ = \{1, 2, 3, \dots\}$. An *encoding function into n bitstrings* is a bijective function mapping $\mathbb{Z}_{2^n} = \{0, 1, \dots, 2^n - 1\}$ to $\{0, 1\}^n$. □

Let $N \leq 2^n$.

- For every pair of finite cyclic group G of order N and its generator, there is an encoding function σ into n bitstrings such that G is isomorphic to $\mathbb{Z}_N$ via σ.
- Conversely, for every encoding function σ into n bitstrings, by defining the binary operation $\sigma(x) \circ \sigma(y) := \sigma(x + y)$ on $\sigma(\mathbb{Z}_N)$, the set $\sigma(\mathbb{Z}_N)$ becomes a finite cyclic group of order N with generator $\sigma(1)$ and the set $\sigma(\mathbb{Z}_N)$ is isomorphic to $\mathbb{Z}_N$ via σ.

In this manner, there is a bijective correspondence between a pair of a finite cyclic group G of order N and its generator, and an encoding function σ into n bitstrings.

By choosing σ appropriately, any finite cyclic group G (with its generator) can be represented.

Generic Algorithm

Definition [Generic Algorithm, Shoup 97]

A *generic algorithm* is a probabilistic oracle Turing machine $\mathcal{A}$ which behaves as follows:

Let $n \in \mathbb{N}^+$, and let σ be an encoding function into n bitstrings and N a positive integer with $N \leq 2^n$.

(i) $\mathcal{A}$ takes as input a list $\sigma(x_1), \ldots, \sigma(x_k)$ with $x_1, \ldots, x_k \in \mathbb{Z}_N$, as well as (the binary representations of) N and its prime factorization.

(ii) As $\mathcal{A}$ is executed, it is allowed to make calls to oracles which compute the functions $add\colon \sigma(\mathbb{Z}_N) \times \sigma(\mathbb{Z}_N) \to \sigma(\mathbb{Z}_N)$ and $inv\colon \sigma(\mathbb{Z}_N) \to \sigma(\mathbb{Z}_N)$ with

$$add(\sigma(x), \sigma(y)) = \sigma(x + y) \quad \text{and} \quad inv(\sigma(x)) = \sigma(-x).$$

The algorithm $\mathcal{A}$ do not perform these operations internally by itself.

(iii) Eventually, $\mathcal{A}$ halts and outputs a finite binary string, denoted by

$$\mathcal{A}(N; \sigma(x_1), \ldots, \sigma(x_k)).$$

□

The Discrete Logarithm Problem in the Generic Group Model

Experiment for the Discrete Logarithm Problem A

Consider the following experiment defined for a polynomial-time generic algorithm $\mathcal{A}$, a parameter n, and a positive integer $N \leq 2^n$:

The discrete logarithm experiment $\mathrm{DLog}_{\mathcal{A}}(n, N)$**:**

1. *Generate an encoding function σ into n bitstrings uniformly.*
2. *Generate $x \in \mathbb{Z}_N$ uniformly.*
3. *The output of the experiment is defined to be* 1 *if*

$$\mathcal{A}(N; \sigma(1), \sigma(x)) = x$$

$\sigma(1)$ is a generator of the finite cyclic group $\sigma(\mathbb{Z}_N)$ of order N, and x is the discrete logarithm of $\sigma(x)$ with respect to $\sigma(1)$.

and 0 *otherwise.*

The Hardness of the Discrete Logarithm Problem A

Theorem [Shoup 97]

There exists $C \in \mathbb{N}^+$ such that, for every generic algorithm $\mathcal{A}$, $n \in \mathbb{N}^+$, and N with $N \leq 2^n$,

$$\mathrm{Prob}[\mathrm{DLog}_{\mathcal{A}}(n, N) = 1] \leq \frac{Cm^2}{p},$$

where p is the largest prime divisor of N and m is the maximum number of the oracle queries among all the computation paths of $\mathcal{A}$. □

If we insist that $\mathcal{A}$ succeed with probability bounded by a positive constant (e.g., 1/2) to the below, this theorem translates into a lower bound $\Omega(\sqrt{p})$ of the number of group operations queried by $\mathcal{A}$.

Translating Shoup's result into the form well used as a computational assumption

Experiment for the Discrete Logarithm Problem B

Consider the following experiment for a polynomial-time generic algorithm $\mathcal{A}$, a parameter n, and an encoding function σ into n bitstrings:

The discrete logarithm experiment $\mathsf{DLog}_{\mathcal{A}}(n, \sigma)$**:**

1. *Generate an* n*-bit prime* p *uniformly.*
2. *Generate* $x \in \mathbb{Z}_p$ *uniformly.*
3. *The output of the experiment is defined to be* 1 *if*

$$\mathcal{A}(p; \sigma(1), \sigma(x)) = x$$

and 0 *otherwise.*

The Hardness of the Discrete Logarithm Problem B

The hardness of the discrete logarithm problem in the generic group model is then formulated as follows.

Definition We say that *the discrete logarithm problem is hard in the generic group model* if for all polynomial-time generic algorithms $\mathcal{A}$ and all $d \in \mathbb{N}^+$ there exists $N \in \mathbb{N}^+$ such that, for all $n > N$,

$$\frac{1}{\#\mathsf{Encf}_n} \sum_{\sigma \in \mathsf{Encf}_n} \mathsf{Prob}[\mathsf{DLog}_{\mathcal{A}}(n, \sigma) = 1] \leq \frac{1}{n^d},$$

where Encf_n is the set of all encoding functions into n bitstrings. □

Note that the probability is averaged over all encoding functions into n bitstrings. This results in a random encoding function into n bitstrings, i.e., the generic group.

Theorem The discrete logarithm problem is hard in the generic group model. □

Our aim is the secure instantiation of the generic group.

For that purpose, we translate Shoup's result into a stronger computational hardness.

To put it plainly, the content of this research is, in essence, to perform computable analysis over cryptography.

The Effective Hardness of the Discrete Logarithm Problem

In this talk we consider a stronger notion of the hardness of the discrete logarithm problem. This stronger notion, called the effective hardness of the discrete logarithm problem, is defined as follows:

We first choose a particular recursive enumeration $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \ldots$ of all polynomial-time generic algorithms. It is easy to show that such an enumeration exists.

The effective hardness of the discrete logarithm problem in the generic group model is then formulated as follows.

Definition We say that *the discrete logarithm problem is effectively hard in the generic group model* if there exists a computable function $f : \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$ such that, for all $i, d, n \in \mathbb{N}^+$, if $n \geq f(i, d)$ then

$$\frac{1}{\#\mathsf{Encf}_n} \sum_{\sigma \in \mathsf{Encf}_n} \mathsf{Prob}[\mathsf{DLog}_{\mathcal{A}_i}(n, \sigma) = 1] \leq \frac{1}{n^d}.$$

□

Effective Hardness ?

In the definitions of the (conventional) hardness of the discrete logarithm problem, the number N is only required to exist, depending on an adversary $\mathcal{A}$ and a number d, that is, the success probability of the attack by an adversary $\mathcal{A}$ on a security parameter n is required to be less than $1/n^d$ for all sufficiently large n, where the lower bound of such n is not required to be computable from $\mathcal{A}$ and d.

On the other hand, in the definitions of the effective hardness of the discrete logarithm problem, it is required that the lower bound N of such n can be computed from the code of $\mathcal{A}$ and d.

Definition [posted again]

We say that *the discrete logarithm problem is hard in the generic group model* if for all polynomial-time generic algorithms $\mathcal{A}$ and all $d \in \mathbb{N}^+$ there exists $N \in \mathbb{N}^+$ such that, for all $n > N$,

$$\frac{1}{\#\mathsf{Encf}_n} \sum_{\sigma \in \mathsf{Encf}_n} \mathrm{Prob}[\mathrm{DLog}_{\mathcal{A}}(n, \sigma) = 1] \leq \frac{1}{n^d}.$$

□

Effective Hardness ?

In modern cryptography based on computational security, it is important to choose the security parameter n of a cryptographic scheme as small as possible to the extent that the security requirements are satisfied, in order to make the efficiency of the scheme as high as possible.

For that purpose, it is desirable to be able to calculate a concrete value of N, given the code of $\mathcal{A}$ and d, since N gives a lower bound of the security parameter for which the security requirements specified by $\mathcal{A}$ and d are satisfied. This results in the notion of effective hardness.

Definition [posted again]

We say that *the discrete logarithm problem is hard in the generic group model* if for all polynomial-time generic algorithms $\mathcal{A}$ and all $d \in \mathbb{N}^+$ there exists $N \in \mathbb{N}^+$ such that, for all $n > N$,

$$\frac{1}{\#\mathsf{Encf}_n} \sum_{\sigma \in \mathsf{Encf}_n} \mathsf{Prob}[\mathsf{DLog}_{\mathcal{A}}(n, \sigma) = 1] \leq \frac{1}{n^d}.$$

□

The Effective Hardness of the Discrete Logarithm Problem

Definition [posted again]

We say that *the discrete logarithm problem is effectively hard in the generic group model* if there exists a computable function $f : \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$ such that, for all $i, d, n \in \mathbb{N}^+$, if $n \geq f(i, d)$ then

$$\frac{1}{\#\mathsf{Encf}_n} \sum_{\sigma \in \mathsf{Encf}_n} \mathsf{Prob}[\mathsf{DLog}_{\mathcal{A}_i}(n, \sigma) = 1] \leq \frac{1}{n^d}.$$

□

Shoup's result can be translated into the following stronger form:

Theorem The discrete logarithm problem is effectively hard in the generic group model. □

Applying algorithmic randomness together with the effective hardness, we securely instantiate the generic group by a computable function.

Application of Algorithmic Randomness

Lebesgue Measure on Families of Encoding Functions

Encf_n: The set of all encoding functions σ into n bitstrings.

Encf^∞: The set of all families of encoding functions, i.e.,

$$\mathsf{Encf}^\infty := \prod_{k=1}^{\infty} \mathsf{Encf}_k = \mathsf{Encf}_1 \times \mathsf{Encf}_2 \times \mathsf{Encf}_3 \times \cdots\cdots .$$

Encf^*: The set of all finite families of encoding functions, i.e.,

$$\mathsf{Encf}^* := \bigcup_{n=0}^{\infty} \left(\prod_{k=1}^{n} \mathsf{Encf}_k \right) .$$

$\mathcal{L}$: Lebesgue measure on Encf^∞

Theorem [generalization of Exercise 1.9.21 of Nies's textbook] Let S be an r.e. subset of Encf^*. Suppose that $\mathcal{L}\left([S]^{\prec}\right) < 1$ and $\mathcal{L}\left([S]^{\prec}\right)$ is a computable real. Then there exists a computable family of encoding functions which is not in $[S]^{\prec}$. □

Secure Instantiation of the Generic Group

Secure Instantiation by computable Function

The hardness of the discrete logarithm problem relative to a specific family of encoding functions is defined as follows.

Definition Let $\{\sigma_n\}_{n\in\mathbb{N}^+}$ be a family of encoding functions. We say that *the discrete logarithm problem is hard relative to* $\{\sigma_n\}_{n\in\mathbb{N}^+}$ if for all polynomial-time generic algorithms $\mathcal{A}$ and all $d \in \mathbb{N}^+$ there exists $N \in \mathbb{N}^+$ such that, for all $n > N$,

$$\text{Prob}[\text{DLog}_{\mathcal{A}}(n, \sigma_n) = 1] \leq \frac{1}{n^d}.$$

□

Theorem [Main Result] There exists a computable family of encoding functions relative to which the discrete logarithm problem is effectively hard. □

Furure Direction

It would be challenging to prove the following conjecture (or its appropriate modification) with identifying an appropriate computational assumption COMP which seems weaker than the hardness of the discrete logarithm problem itself. Here the notion of effective hardness is replaced by the notion of polynomial-time effective hardness.

Conjecture

Under the assumption COMP, there exists a polynomial-time computable family of encoding functions (or a polynomial-time computable family of families of encoding functions) relative to which the discrete logarithm problem is polynomial-time effectively hard. □

The conjecture states that the discrete logarithm problem is hard in the standard model for some polynomial-time computable finite cyclic group.